Any questions or comments please contact Douglass George in Rm 35

Phone: 918-7488

E-mail dgeorge@v2kmail.gsfc.nasa.gov

Vision 2000 CCS
System Monitor's
Manage Events
Application Programming Interface
Developers Guide
Release 2.1
(2-13-97)

Any questions or comments please contact Douglass George in Rm 35

Phone: 918-7488

E-mail dgeorge@v2kmail.gsfc.nasa.gov

Functional Description

For release 2.1, a Manage Events (EVT) Application Programmer Interface (API) will be provided for C++ applications running on the IRIX 6.2 operating system. The EVT API will use CCS Middleware techniques for interprocess communication (IPC), particularly the ability to pass objects between processes. The developer will have to perform the following activities to successfully use the EVT API:

1) The developer of a CCS application using this API will have to include the files listed below into their source code:

SYM_EvtGenerator.h

2) Developers will also have to link in with the following libraries to include this API.

l ibevtgen.so

3) Developers will need to set the following environment variables in their run scripts. These environment variables are retrieved by the EVT API code when CCSDEBUG or UNITTEST flags are set in the Makefile. These environment variables should then be removed or commented out when not in CCSDEBUG or UNITTEST mode.

SYM_EVT_LOG_DIR	set to the directory path for output logs
SYM_EVT_LOG_FILE	set to a meaningful filename
SYM_EVT_TEST_BIN	set to a meaningful filename
SYM_EVT_NETMODE	set to 1 or 2

4) It is necessary to provide the DMG team a process name for every deliverable class (application.) This process name will be entered into the oracle database with a corresponding Process Name ID number. It is also necessary to provide DMG with an event type, event severity, and event background that will depend on an event ID. These values will be put into the data base for later use.

The event type and event severity enumerated types are provided below.

```
enum SYM_EvtEnumType {

SYM_EvtTypeNONE,                    //-- PRS Legacy Event Type
SYM_EvtTypeTELEMETRY,               //-- PRS Legacy Event Type
SYM_EvtTypePLAYBACK,                //-- PRS Legacy Event Type
SYM_EvtTypeREALTIME,                //-- PRS Legacy Event Type
SYM_EvtTypeNCC,                     //-- PRS Legacy Event Type
SYM_EvtTypeGROUND_CONFIGURATION,    //-- PRS Legacy Event Type
SYM_EvtTypeALARM,                   //-- PRS Legacy Event Type
SYM_EvtTypeCOMMAND_DIALOG,          //-- PRS Legacy Event Type
SYM_EvtTypePSTOL_RESPONSE,          //-- PRS Legacy Event Type
SYM_EvtTypeGENERAL,                 //-- PRS Legacy Event Type
SYM_EvtTypeSYSTEM_SOFTWARE_ERROR,   //-- PRS Legacy Event Type
SYM_EvtTypeKEYBOARD_INPUT_ECHO,     //-- PRS Legacy Event Type
SYM_EvtTypeOFLS_OLS_FILE_TRANSFER,  //-- PRS Legacy Event Type
SYM_EvtTypeCOMMAND_PAGE             //-- PRS Legacy Event Type
```

```
}; //-- end of enumeration definition

enum  SYM_EvtEnumSeverity {

SYM_EvtSeverityDEBUG,         // Debug Mode - Events not archived
SYM_EvtSeverityINFORMATIONAL, // Informational event
SYM_EvtSeverityERROR,         // Error Event - low priority
SYM_EvtSeverityFATAL          // Fatal Event - high priority

}; //-- end of enumeration definition
```

Any questions or comments please contact Douglass George in Rm 35

Phone: 918-7488

E-mail dgeorge@v2kmail.gsfc.nasa.gov

Supporting Documentation

The following documentation provides a detailed description of how event messages are used in the CCS context and explain the terminology used in this EVT API Developer's Guide.

- Manage Events Straw-Person Memorandum
- Vision 2000 Control Center System Monitoring Manage Events Release 2.1 Design Walk-through
- Managing Events Functional Requirements for the Hubble Space Telescope Control Center System

This documentation is available at http://ccs.gsfc.nasa.gov/ccspages/teaminfo/pat/Evt/Ma_evt.htm

Any questions or comments please contact Douglass George in Rm 35

Phone: 918-7488

E-mail dgeorge@v2kmail.gsfc.nasa.gov

Directions for Use

The CCS application must first initialize the API by instantiating the proxy class `SYM_EvtGenerator`

METHOD	**`SYM_EvtGenerator( short unsigned int processNameID, SYM_EvtSubSystem subSystem_);`**
PURPOSE	Constructor
ARGUMENTS	**processNameID** used to determine the name of process or class that is calling this particular proxy object. The integer passed for this parameter will correspond to a textual description of the process name maintained by the DMG subsystem. **subSystem** used to determine the name of the subSystem that is using this particular proxy object. Enumerated types are as follows: CCS_CCS CCS_FEP CCS_CMD CCS_SYM CCS_DMG CCS_CCM CCS_GUI CCS_MDW
RETURN VALUE	None

The CCS application using the EVT API must call a series of method calls. The `SYM_EvtGenerator.createEvent` and `SYM_EvtGenerator.sendEvent` method calls must be called in respective order. In addition, a CCS developer can use any of the overloaded `SYM_EvtGenerator.addForeground` methods, after the `SYM_EvtGenerator.createEvent` method, to add necessary foreground elements to an event message.

Any questions or comments please contact Douglass George in Rm 35

Phone: 918-7488

E-mail dgeorge@v2kmail.gsfc.nasa.gov

METHOD	`short int createEvent( const short unsigned int eventID const SYM_EvtEnumOpMode opMode )`
PURPOSE	To create an event message containing the id number of the message and the operational mode of the calling application.
ARGUMENTS	**eventID** used to identify the id number as indicated in the Event Lookup Table provided by the DMG subsystem **opMode** used to identify the operational mode of the application using this particular proxy object. Enumerated types are as follows: SYM_EvtOpNONE SYM_EvtOpREAL_TIME SYM_EvtOpHISTORICAL
RETURN VALUE	A short integer signifying success (0) or failure (1)

METHOD	`addForeground ( const short int shortParameter )`
PURPOSE	Add a short int foreground element to the event message.
ARGUMENTS	**shortParameter** the short integer type that is to be added as a foreground element to the event message.
RETURN VALUE	A short integer signifying success (0) or failure (1)

METHOD	`addForeground ( const long int longParameter )`
PURPOSE	Add a long int foreground element to the event message.
ARGUMENTS	**longParameter** the long integer type that is to be added as a foreground element to the event message.
RETURN VALUE	A short integer signifying success (0) or failure (1)

Any questions or comments please contact Douglass George in Rm 35

Phone: 918-7488

E-mail dgeorge@v2kmail.gsfc.nasa.gov

METHOD	`addForeground ( const float floatParameter )`
PURPOSE	Add a float foreground element to the event message.
ARGUMENTS	**floatParameter** the float type that is to be added as a foreground element to the event message.
RETURN VALUE	A short integer signifying success (0) or failure (1)

METHOD	`addForeground ( const double doubleParameter )`
PURPOSE	Add a double foreground element to the event message.
ARGUMENTS	**doubleParameter** the double type that is to be added as a foreground element to the event message.
RETURN VALUE	A short integer signifying success (0) or failure (1)

METHOD	`addForeground ( const char* stringParameter )`
PURPOSE	Add a string foreground element to the event message.
ARGUMENTS	**stringParameter** the character array type that is to be added as a foreground element to the event message.
RETURN VALUE	A short integer signifying success (0) or failure (1)

METHOD	`addForeground ( const RWCString rwStringParameter )`
PURPOSE	Add a RWCString foreground element to the event message.
ARGUMENTS	**rwStringParameter** the RWCString type that is to be added as a foreground element to the event message.
RETURN VALUE	A short integer signifying success (0) or failure (1)

Any questions or comments please contact Douglass George in Rm 35

Phone: 918-7488

E-mail dgeorge@v2kmail.gsfc.nasa.gov

METHOD	`short int sendEvent( )`
PURPOSE	To send the newly created event message to the EVT for processing. In debug mode, the events will be stored on a local file in ASCII text format.
ARGUMENTS	None
RETURN VALUE	A short integer signifying success (0) or failure (1)

As an alternative approach is to use the following overloaded operators to "stream" the necessary data of creating and sending event messages. An example of using this technique is provided in the Example section of this document.

OPERATOR	`SYM_EvtGenerator& SYM_EvtGenerator::operator ()`
	`( const unsigned short int& eventID,`
	`const SYM_EvtEnumOpMode& opMode ){`

PURPOSE	To create an event message containing the id number of the message and the operational mode of the calling application.
ARGUMENTS	**eventID** used to identify the id number as indicated in the Event Lookup Table provided by the DMG subsystem **opMode** used to identify the operational mode of the application using this particular proxy object. Enumerated types are as follows: SYM_EvtOpNONE SYM_EvtOpREAL_TIME SYM_EvtOpHISTORICAL
RETURN VALUE	**SYM_EvtGenerator&** a reference to a SYM_EvtGenerator object

OPERATOR	`SYM_EvtGenerator& operator <<`
	`( SYM_EvtGenerator& generator,`
	`const short int& shortParameter){`

PURPOSE	Add a short int foreground element to the event message.
ARGUMENTS	**SYM_EvtGenerator&** a reference to a SYM_EvtGenerator object **shortParameter** the short integer type that is to be added as a foreground element to the event message.
RETURN VALUE	**SYM_EvtGenerator&** a reference to a SYM_EvtGenerator object

Any questions or comments please contact Douglass George in Rm 35

Phone: 918-7488

E-mail dgeorge@v2kmail.gsfc.nasa.gov

OPERATOR	**`SYM_EvtGenerator& operator <<`** **`( SYM_EvtGenerator& generator,`** **`  const long int& longParameter )`**
PURPOSE	Add a long int foreground element to the event message.
ARGUMENTS	**SYM_EvtGenerator&** a reference to a SYM_EvtGenerator object **longParameter** the long integer type that is to be added as a foreground element to the event message.
RETURN VALUE	**SYM_EvtGenerator&** a reference to a SYM_EvtGenerator object

OPERATOR	**`SYM_EvtGenerator& operator <<`** **`( SYM_EvtGenerator& generator,`** **`  const float& floatParameter ){`**
PURPOSE	Add a float foreground element to the event message.
ARGUMENTS	**SYM_EvtGenerator&** a reference to a SYM_EvtGenerator object **floatParameter** the float type that is to be added as a foreground element to the event message.
RETURN VALUE	**SYM_EvtGenerator&** a reference to a SYM_EvtGenerator object

OPERATOR	**`SYM_EvtGenerator& operator <<`** **`( SYM_EvtGenerator& generator,`** **`  const double& doubleParameter ){`**
PURPOSE	Add a double foreground element to the event message.
ARGUMENTS	**SYM_EvtGenerator&** a reference to a SYM_EvtGenerator object **doubleParameter** the double type that is to be added as a foreground element to the event message.
RETURN VALUE	**SYM_EvtGenerator&** a reference to a SYM_EvtGenerator object

Any questions or comments please contact Douglass George in Rm 35

Phone: 918-7488

E-mail dgeorge@v2kmail.gsfc.nasa.gov

METHOD	`addForeground ( const char* stringParameter )`

OPERATOR	`SYM_EvtGenerator& operator <<`
	`( SYM_EvtGenerator& generator,`
	`  const char* stringParameter ){`

PURPOSE	Add a string foreground element to the event message.
ARGUMENTS	**stringParameter** the character array type that is to be added as a foreground element to the event message.
RETURN VALUE	A short integer signifying success (0) or failure (1)

OPERATOR	`SYM_EvtGenerator& operator <<`
	`( SYM_EvtGenerator& generator,`
	`  const RWCString& rwStringParameter ){`

PURPOSE	Add a RWCString foreground element to the event message.
ARGUMENTS	**SYM_EvtGenerator&** a reference to a SYM_EvtGenerator object **rwStringParameter** the RWCString type that is to be added as a foreground element to the event message.
RETURN VALUE	**SYM_EvtGenerator&** a reference to a SYM_EvtGenerator object

OPERATOR	`void <<`
	`( SYM_EvtGenerator& generator,`
	`  const SYM_EvtEnumSend& aSend){`

PURPOSE	To send the newly created event message to the EVT for processing. In debug mode, the events will be stored on a local file in ASCII text format.
ARGUMENTS	**SYM_EvtGenerator&** a reference to a SYM_EvtGenerator object **aSend** is an enumerated type that must be SYM_EvtSEND (case sensitive.)
RETURN VALUE	**void** no return value

It is important to keep the data types in a form the expert system applications can use (for example: keeping the value '6' as an short int and not as a char or string). This will ensure that expert system applications can perform inferencing using the short int, long int, float, and double data types of these event messages. The number of elements and each element's corresponding data type may be found in the Event Lookup table provided by the DMG subsystem.

Any questions or comments please contact Douglass George in Rm 35

Phone: 918-7488

E-mail dgeorge@v2kmail.gsfc.nasa.gov

CCS Applications will call the following destructor when shutting down:

METHOD	**~SYM_EvtGenerator()**
PURPOSE	Destructor
ARGUMENTS	None
RETURN VALUE	None

Any questions or comments please contact Douglass George in Rm 35

Phone: 918-7488

E-mail dgeorge@v2kmail.gsfc.nasa.gov

Example

Scenario

The examples listed below have hard-coded values. It is strongly recommended that developers avoid the use of hard-coded values by using constants, environment variables, etc.

Suppose a SYM developer, named Doug, was finishing up his application named "Fault Detection Request Manager." This application wanted to generate an event message every time his application was initialized. Doug has already entered in the message ID, Severity, Type, and Background into the Event Lookup Table via the DMG provided interface. The event message Doug wants to create contains the following information in the Event Lookup table:

ID_NUMBER	30,023
SEVERITY	Informational
TYPE	Software State Change
BACKGROUND	Process started with %d requests by user %s

Example 1. Using overloaded methods:

Doug will then make sure he has instantiated a `SYM_EvtGenerator` object in his initialization routine:

```
.
.
.
unsigned short  int          myProcessNameID  = 67; // The corresponding
                             //process name "FLT DT REQ MGR
                             //will be displayed by the GUI.
SYM_EvtSubSystem       mySubSystem      = CCS_SYM;//The corresponding
                             //subsystem name "SYM" will be displayed
                             // by the GUI
.
.
.
SYM_EvtGenerator  myGenerator(  myProcessNameID,  // add process name
                                mySubSystem )   // add subsystem
.
.
.
```

Doug will then make sure he has called the `SYM_EvtGenerator.createEvent` method correctly.

```
.
.
.
SYM_EvtOpMode    myCurrentMode    = SYM_EvtOpREAL_TIME;
int              currentRequests  = 6;
char*            myUser           = "DOUG";
.
.
.
myGenerator.createEvent( 30023,             // set     eventID
                         myCurrentMode )    // set   opmode
```

```
.
.
.
```

Doug will then make sure he add the Foreground elements to the event message.

```
myGenerator.addForeground(   currentRequests ) // add a short    int 6 to
                                                //the Foreground
myGenerator.addForeground(   myUser ) //add string "DOUG" to the
                                      //Foreground
```

Doug will then make sure he completes the series of calls by sending the event message.

```
myGenerator.sendEvent( ) //  sends the message to EVT for processing

.
.
.
```

Doug runs his application and sees the following on the Event Analyzer Display:

1996/365/16:33:32.000 SYM10023 I SSC "FLT DT REQ MGR" BB "PROCESS STARTED WITH 6 REQUESTS BY USER DOUG"

Example 2. Using overloaded operators:

Doug will then make sure he has instantiated a `SYM_EvtGenerator` object in his initialization routine:

```
.
.
.
unsigned short  int          myProcessNameID  = 67; // The corresponding
                             //process name "FLT DT REQ MGR
                             //will be displayed by the GUI.
SYM_EvtSubSystem       mySubSystem      = CCS_SYM;//The corresponding
                             //subsystem name "SYM" will be displayed
                             // by the GUI
.
.
.
SYM_EvtGenerator  myGenerator(  myProcessNameID,  // add process name
                                mySubSystem )   // add subsystem
.
.
.
```

Doug will then make sure he has called the `SYM_EvtGenerator.createEvent` method correctly.

```
.
.
.
SYM_EvtOpMode    myCurrentMode    = SYM_EvtOpREAL_TIME;
int              currentRequests  = 6;
char*            myUser           = "DOUG";
.
.
.
myGenerator.createEvent( 30023,
```

```
                              myCurrentMode )
myGenerator(30023,             // set    eventID
             myCurrentMode ) // set   opmode
<< currentRequests     // add a short    int 6 to the Foreground
<< myUser              //add string "DOUG" to the
<< SYM_EvtSEND;        //  sends the message to EVT for processing
.
.
.
```

Doug runs his application and sees the following on the Event Analyzer Display:

1996/365/16:33:32.000 SYM10023 I SSC "FLT DT REQ MGR" BB "PROCESS STARTED WITH 6 REQUESTS BY USER DOUG"